BBN Systems and Technologies
A Division of Bolt Beranek and Newman Inc.

1

BBN Report No. 7622

NETWORK INTERFACE UNIT (NIU)

DETAILED DESIGN SPECIFICATION

bbn

REPORT DOCUMENTATION PAGE

Form Approved OPM No. 0704-0188

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE June 1991	3. REPORT TYPE AND DATES COVERED Detailed Design Specification

4. TITLE AND SUBTITLE Network Interface Unit (NIU) Detailed Design Specification	5. FUNDING NUMBERS Contract Numbers: MDA972-89-C-0060 MDA972-89-C-0061
6. AUTHOR(S) Author unspecified.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Bolt Beranek and Newman, Inc. (BBN) Systems and Technologies; Advanced Simulation 10 Moulton Street Cambridge, MA 02138	8. PERFORMING ORGANIZATION REPORT NUMBER BBN Report Number: 7622
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency (DARPA) 3701 North Fairfax Drive Arlington, VA 22203-1714	10. SPONSORING/MONITORING AGENCY REPORT NUMBER DARPA Report Number: None
11. SUPPLEMENTARY NOTES None	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Distribution Statement A: Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE Distribution Code: A

13. ABSTRACT *(Maximum 200 words)*

A Simulation Network (SIMNET) project detailed design specification describing the Network Interface Unit (NIU) software of the SIMNET hardware and software training system for vehicle crew training and operational training.

14. SUBJECT TERMS The SIMNET Network Interface Unit (NIU) software detailed design specification.	15. NUMBER OF PAGES
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT Same as report.

NSN 7540-01-280-5500

Standard Form 298, 880922
Prescribed by ANSI Std. 239-18
299-01

Report No. 7622

NETWORK INTERFACE UNIT (NIU) DETAILED DESIGN SPECIFICATION

June 1991

Prepared by:

BBN Systems and Technologies
Advanced Simulation
10 Moulton Street
Cambridge, MA 02138 USA

Prepared for:

Defense Advanced Research Projects Agency (DARPA)
1400 Wilson Blvd.
Arlington, VA 22209-2308

This research was performed by BBN Systems and Technologies under Contract No. MDA972-90-C-0061 to the Defense Advanced Research Projects Agency (DARPA). The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policy, either expressed or implied, of DARPA, the U.S. Army, or the U.S. Government.

Table of Contents

SECTION 1. GENERAL

1.1 Purpose of the NIU Detailed Design Specification. The NIU Detailed Design Specification has been written to fulfill the following objectives:

a. To provide a detailed definition of the NIU functions.
b. To discuss NIU timing.
c. To identify the NIU design goals and the means in which they have been met.

1.2 Project References. The following documents are recommended to provide background information on the NIU, and SIMNET.

BBN Report No. 7102 July 1989	The SIMNET Network and Protocols
The SIMNET Communications Protocol For Distributed Simulation	
The VMEbus Specification rev C.1	

1.3 Terms and Abbreviations.

ICD	**Interface Control Document** A document describing an interface between two devices, in this case the NIU and a host.
NIU	**Network Interface Unit** Connects a simulator to a SIMNET network.
RVA	**Remote Vehicle Approximation** Dead reckoning algorithm used to reduce network traffic. Currently first-order extrapolation.
PDU	**Protocol Data Unit** A SIMNET message. Packets are groups of PDUs which are encapsulated. The PDU is the SIMNET unit of network data.
DVG	**Digital Voice Gateway** Software which encodes voice digitally for transmission over a network.

SOW	**State of the World** The NIU's view of the states of other simulators on the network.
SIMNET	**Simulator Network** A DARPA network for connecting combat simulators.
CIG	**Computer Image Generator** A BBN image generator which has been integrated to the NIU.
VAP	**Vehicle Appearance PDU** A SIMNET PDU which contains vehicle appearance data.

SECTION 2. NIU OVERVIEW

2.1 NIU Desciption. The NIU provides the capability to connect dissimilar simulators to an existing SIMNET network. The NIU transforms the existing simulator data into SIMNET PDUs, and provides additional overhead required to support the SIMNET protocol. The NIU allows simulators with non-homogeneous frame times to be interoperable on the same network.

→ See #RDP ([illegible])

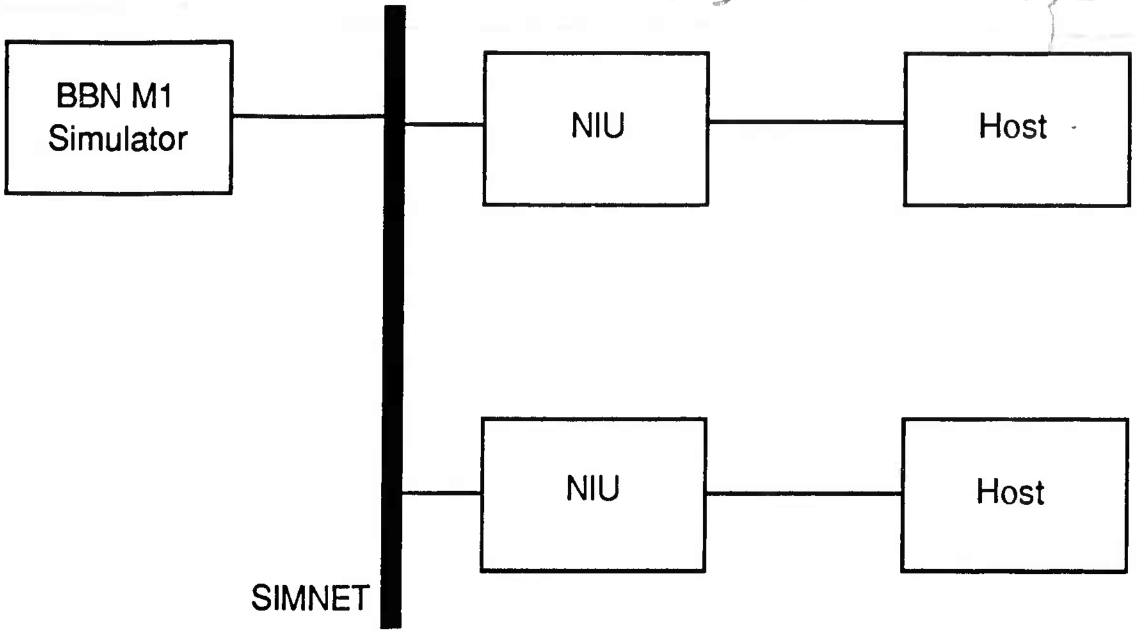


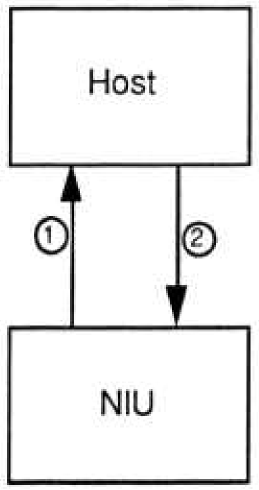
Figure 1. Relationship Between the NIU, Host, and SIMNET Network

2.2 Functions of the NIU. The NIUs main function is the translation of data from host specific interface protocols to SIMNET PDUs, and back. This involves both the repackaging of information in different data-structures, as well as the conversion of units and coordinate systems. The NIU also performs Remote Vehicle Approximation (RVA) which requires the NIU to maintain low-fidelity models of other vehicles on the network. The network transmits updated information on vehicle attitude and position only when the low-fidelity models deviate from the actual model by a pre-defined threshold. The NIU also provides host-specific filtering functions to prevent network traffic from overloading the host.

2.2.1 Accuracy and Validity. The NIU performs all transformations of data such that the precision of the result is the same as the smaller of the operands.

__2.2.2 Timing__. The principle measure of time on the NIU is the frame. NIU frame lengths are either based on transmissions from the host or are based on an internal frame timer (in the event the NIU is self-synced). Self-syncing is used for host interfaces which do not transmit host_to_niu buffers to the NIU, this allows the host to receive regular updates from the NIU without requiring the host to provide any synchronization information. For interfaces which exchange data between the host and NIU, self-syncing will result in a distortion of the RVA's SOW (See **4.2.4.4** for an explaination of this).

Throughout this document buffers which are exchanged between the host and NIU are, for clarity, labeled either "host_to_niu" or "niu_to_host". An exchange will have the host sending the host_to_niu buffer followed immediately by the NIU sending the niu_to_host buffer. Figure **TBD** shows this exchange.

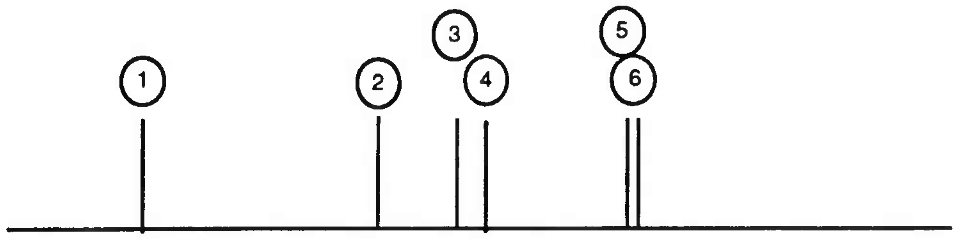


① niu_to_host message buffer

② host_to_niu message buffer

Figure 2.1 Host/NIU Buffer Exchanges

Each NIU on the network is making buffer exchanges with its host once per frame. The timing diagram in Figure **TBD** shows a weapon launch from one host as it propogates through the network to a second host. Since there is no synchronization between NIU's (and therefore between the NIU's hosts) the worst case timing data is shown. The average latency between events 1&2 and 4&5 should be 1/2 of a frame,
assuming a random distribution of events, host frame times and phase differences between hosts.



① Launch occurs

② Launch data passed to NIU (with other data from host)

③ NIU issues SIMNET Fire PDU

④ NIU' receives Fire PDU

⑤ Host' passes data to NIU'

⑥ NIU' passes data to Host' (including Fire data from ②)

Time:

① to ②	Host frame time (worst case)
② to ③	NIU processing time **TBD** (worst case)
③ to ④	Indeterminate; nominally **TBD** for normal operation
④ to ⑤	Host' frame time (worst case)
⑤ to ⑥	Nominally **TBD** (worst case)

Figure 2.2 Timing of Event Propogation Through Network

After the launch occurs in event 1 the host must wait until the start of a frame (or a uniform offset into its own frame) to send this data to the NIU. The worst case would be a launch occuring immediately after a data transfer to the NIU. The NIU next needs to convert the weapon launch data into a SIMNET Fire PDU and write it out on the network. The delay between events 3 and 4 is the transport delay through the SIMNET network, this is affected by network traffic volume and the topology of the network itself. The data shown in Figure TBD was measured at Armstrong Labs-AZ with 2 vehicles on an otherwise unloaded network. After receiving the Fire PDU from SIMNET, NIU' converts it into a host-specific format; it then must wait until the start of the next frame before it may send the data to the host. The delay between events 5 and 6 is the time it take the NIU to receive the host transmission and transmit the niu_to_host buffer.

Vehicle appearance data (either a VAP or host message) is handled differently than events. Apperance data is used only to update the NIU's SOW (See **4.3.1 Packet Flow**), therefore the above timing data is applicable to appearance data only for NIU/Host communications. The frequency of communications over SIMNET is dependant on the RVA algorithm, host vehicle behavior, and RVA thresholds. (For an explaination of this see **4.2.4.3 RVA Update** and **4.4.1.4 Remote Vehicle Approximation**). To date there has been an observed transmission rate of **TBD%** of the host's transmissions for an arbitrarily chosen exercise.

2.3 Flexibility. The NIU may be easily modified to support new host interface protocols, the modification of an existing host interface protocol, and extensions to SIMNET.

SECTION 3. ENVIRONMENT

3.1 Hardware. The NIU consists of a 19" rack mountable VMEbus chassis with two Motorola MVME147 CPUs, two MVME712 transition boards, a CMC ENP100 Ethernet board, a BBN SIMVAD digital audio board, and a mass storage unit with a 150MB SCSI disk and a 150MB SCSI tape drive.

Each MVME147 is connected to a MVME712 which provides an ethernet port, 4 RS232 serial ports, a 8-bit Centronics parallel port, and a SCSI port. The NIU requires a VT100 compatible terminal to be connected to the first CPU's console port at 9600 bps.

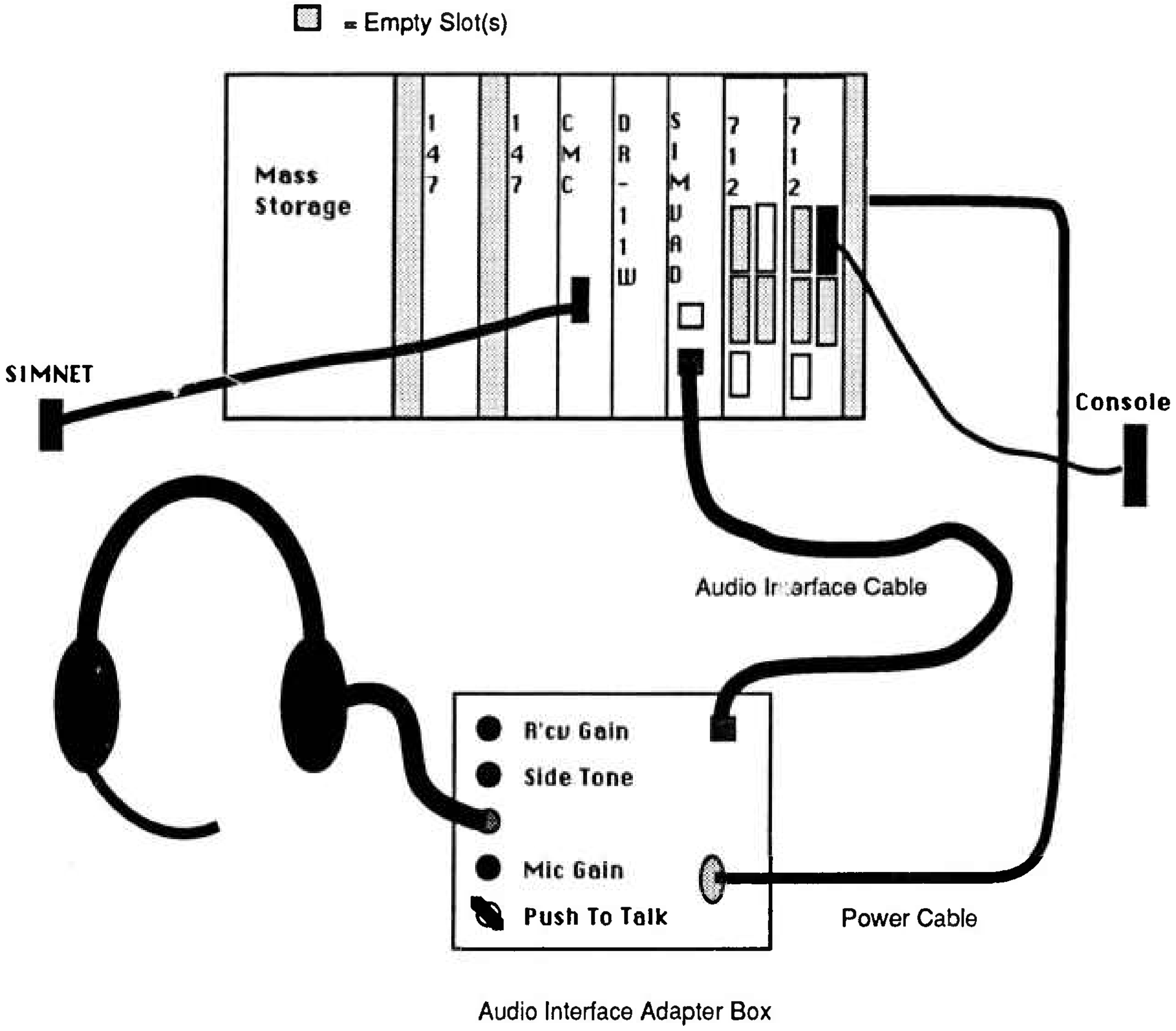


Figure 2. NIU Cable Connections

3.2 Support Software. The NIU software runs under the BBN's GTOS operating system, which was built on top of Ready System's VRTX operating system. Both GTOS and the NIU software are installed on the NIU's 150MB disk drive.

The NIU software is written in the C programming language, and is compiled on a Sun workstation using the C compiler Sun distributes with SunOS 4.0.3. Object files and libraries are linked using the linkage editor distributed with SunOS 4.0.3.

Digital Audio is currently being supported using BBN's Digital Voice Gateway (DVG) software. This is also maintained on a Sun Workstation using identical development tools.

3.3 Interfaces. The NIU is designed to minimize the impact on existing simulators. The physical interface to these simulators is logically separate from the rest of the NIU, and can easily be changed.
Currently several interfaces are supported:

a. Ethernet
b. DR11W
c. HSD

The interface protocol used to communicate with the host is documented in a separate Interface Control Document (ICD).

3.4 Files. Static information about the NIU configuration is kept in several files on the NIU:

/simnet/niu/bin/load_name	The NIU software described here
/simnet/niu/data/nithresh.d	RVA thresholds
/simnet/niu/data/niuprlst.d	NIU range limit for targets
/simnet/data/assoc.def	The host's SIMNET address

SECTION 4. DESIGN DETAILS

4.1 General Operating Procedures. Detailed operating instructions are provided in the Network Operation Procedures.

4.2 Logical Control Flow. Control of program flow in the NIU is regulated by the Simulation State Machine which sequences states in the order shown in Figure 3. The predominant operational state of the NIU is the Simulate State.

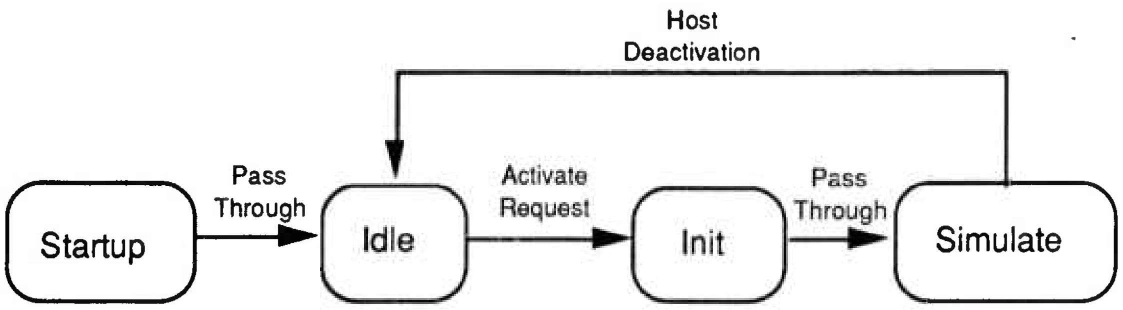


Figure 3. Simulation State Machine States

4.2.1 The Startup State. The Startup state is where initialization is performed of those things which only need to be initialized once, during boot. For example, NIU statistics, timers, and the network.

4.2.2 The Idle State. The Idle state waits for an activation request to be received for the host.

4.2.3 The Init State. The Init state performs per-run initialization of those things needing to be reset after each exercise, such as buffers, thresholds, and RVA data structures.

4.2.4 The Simulate State. The Simulate state is where the translation and Remote Vehicle Approximation is performed. The NIU remains in the Simulate state until the host vehicle is deactivated, at which time it returns to the Idle state. This is the predominate operational state of the NIU. A detail of this state is shown in
Figure 4.

At the start of a frame, the previous frame's data is sent to the host, and the new host buffer is processed. Then RVA-ed vehicle update data is placed in a new message buffer to be transmitted at the start of the next frame. Next the dead-reckoning models of other SIMNET vehicles are updated. Finally the remainder of the current frame is spent polling both SIMNET and the host network. Any SIMNET PDUs received are processed, possibly causing new messages to be appended to the host's message buffer. The frame ends when either a packet is received from the host, or if we are self-synched when the frame timer expires.

4.2.4.1 Processing of the Host Message Buffer. A frame begins with the receipt of the host_to_niu message buffer (inicated by the Frame Event in Figure **TBD**) from the host, so the first activity to occur (following the transmission of an niu_to_host message buffer) is the processing of that buffer. The buffer is in a host-dependant format, and may require more than one physical packet to transmit. (The concept of multiple non-related messages passed from host to NIU is common to all currently existing host interfaces)

A processing function exists in the NIU for each message type which the host will pass to the NIU. The NIU passes a copy of each message extracted from the host_to_niu buffer to the processing function, which then either transforms the message into one or more SIMNET PDUs, or updates the NIU's database so that the information can be incorporated into a PDU at a later time (RADAR data, for example).

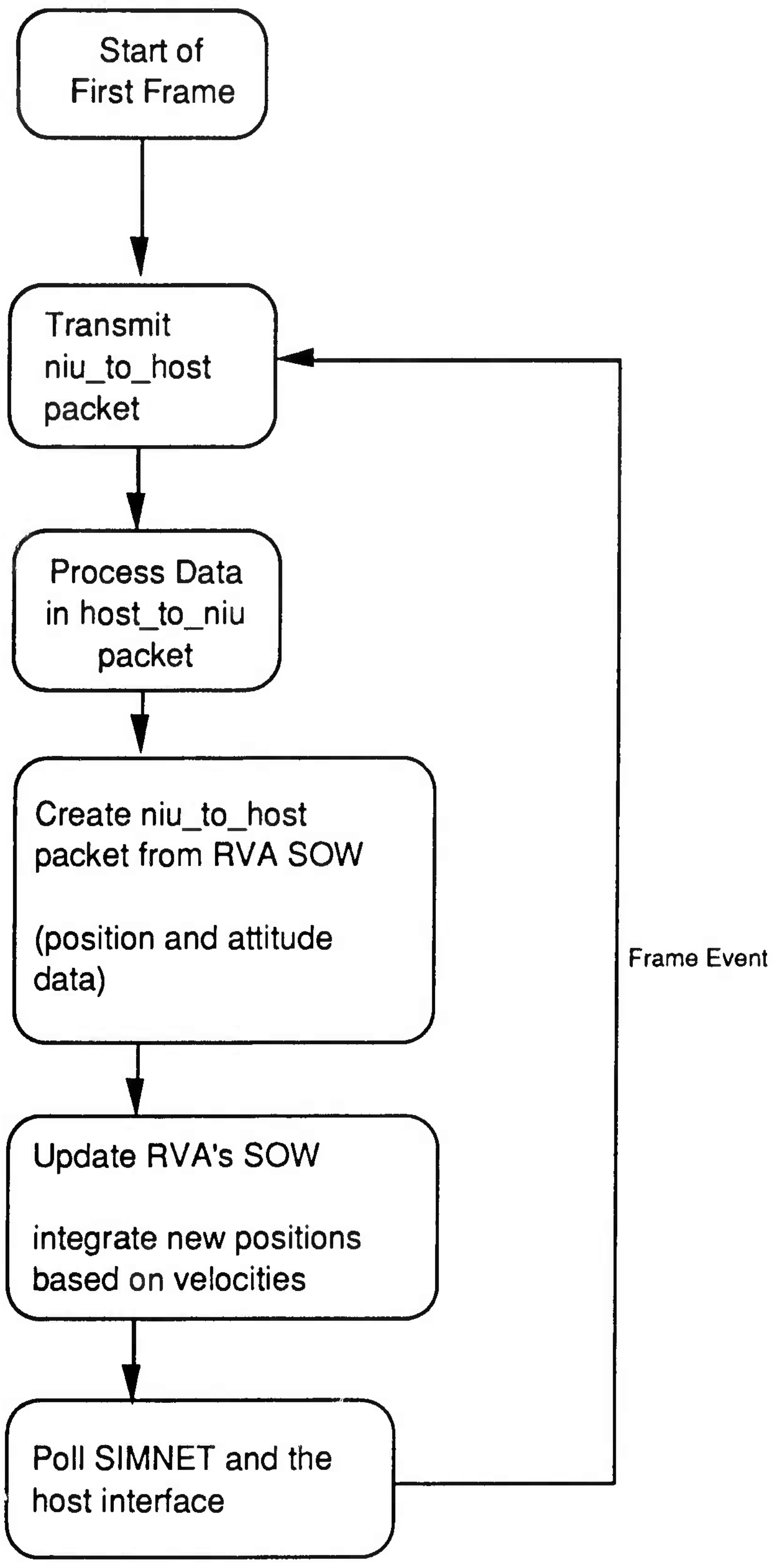

Start of First Frame
Transmit niu_to_host packet
Process Data in host_to_niu packet
Create niu_to_host packet from RVA SOW
(position and attitude data)
Update RVA's SOW
integrate new positions based on velocities
Poll SIMNET and the host interface
Frame Event

Figure TBD. Detail of the Simulate State

4.2.4.2 Creation of the Host Message Buffer. After the host_to_niu buffer has been processed, information on SIMNET vehicles is entered into the niu_to_host buffer which will be transmitted at the start of the next NIU frame. This information is derived from the RVA models, and is within the thresholds placed in nithresh.d. Messages to the host regarding events will be appended to the niu_to_host buffer when SIMNET PDUs are read and processed.

4.2.4.3 RVA Update. One of the NIU's major functions is to perform RVA in order to minimize the modifications to existing simulators before they may operate in a SIMNET exercise. Remote Vehicle Approximation is a dead-reckoning process by which a vehicle's position is extrapolated from a previously known position and velocity vector. Each simulator maintains a dead reckoning model of other vehicles on the network (remote vehicles) as well as a dead reckoning model of itself (and any other vehicles it may be simulating) in the World Coordinates reference system. The simulator compares its present state with the dead reckoning model to determine if the model has deviated by a predefined threshold, and if so transmits an update over the network. This update is used by other simulations to adjust their dead reckoning models, which will be used to determine vehicle positions.

In this manner network traffic is reduced to an extent dependant on the dead-reckoning algorithm, thresholds and vehicle flight. This requires that each simulator on the network RVA other SIMNET vehicles, and that each simulator is using the same RVA algorithm. Currently a linear extrapolation of position is used for all SIMNET simulators (see figure **TBD**). Other RVA algorithms may be included in the future, such as second order position dead reckoning, and/or linear attitude dead reckoning. Experimentation has demonstrated both of these to have a significant impact on the amount of network traffic generated.

$$P_{xnew} = P_{xold} + V_x * dt$$
$$P_{ynew} = P_{yold} + V_y * dt$$
$$P_{znew} = P_{zold} + V_z * dt$$

where P_{*new} = new position component c ; (c = x,y,z)
P_{*old} = old position component c ; (c = x,y,z)
V_* = velocity component c; (c = x,y,z)
dt = time integration period as measured by NIU internal clock.

Figure TBD Remote Vehicle Approximation method.

4.2.4.4 Polling of SIMNET and the Host Network. The remainder of the frame is spend polling SIMNET and the host interface (to check for a new host_to_niu buffer). Figure **TBD** shows the loop the NIU executes until a frame event occurs.

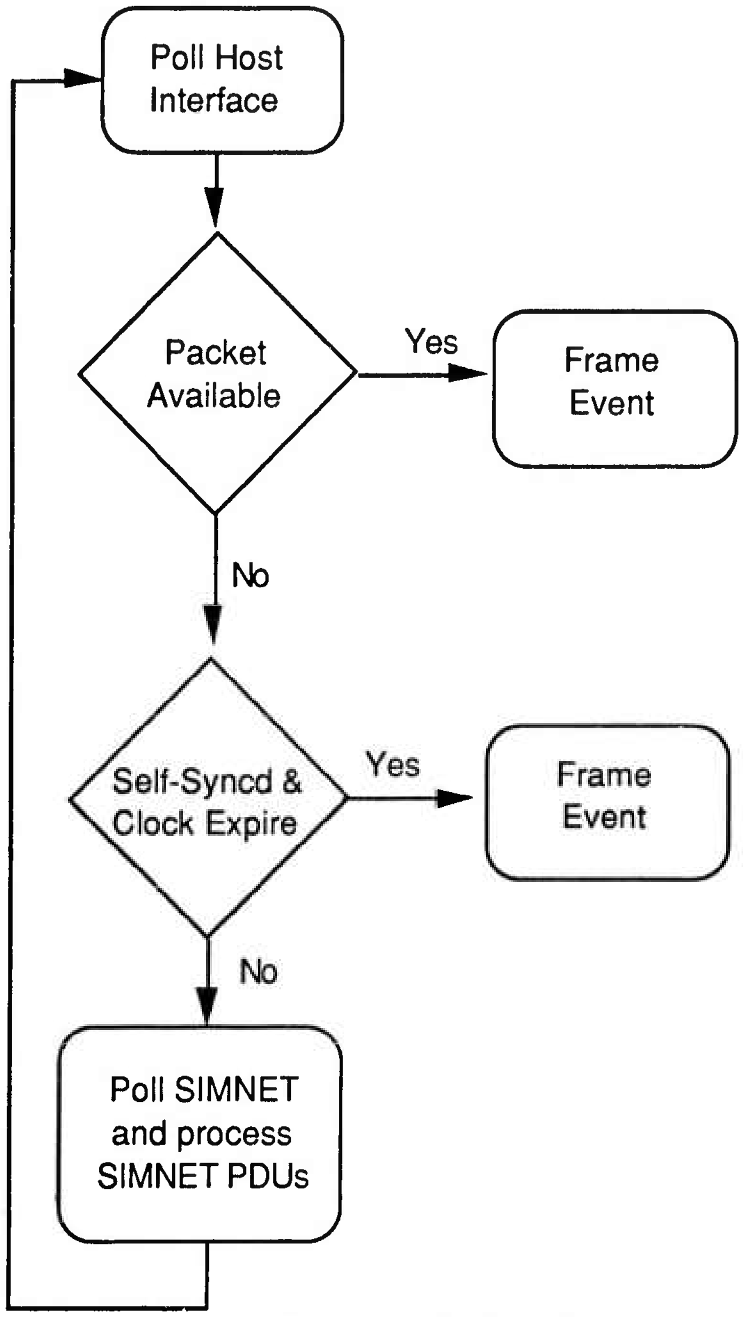


Figure TBD. Polling Loop

A frame event may occur due to a packet arriving from the host, or due to a frame timer expiring if we are self-synced. (Self-synced mode is useful only when there is no incoming traffic from the host to sync off of, otherwise RVA will be distorted due to additional updates).

In the event that the NIU is synced with the host and no packets are received from the host the current NIU frame will never expire.

In the event that the packets from the host are always available, such as would occur with a constantly transmitting host, the NIU will never read or process SIMNET traffic.

The processing of SIMNET PDUs is handled similarly to the way host messages are processed. A processing function exists on the NIU for each SIMNET PDU type, when a PDU is received it is passed to the appropriate processing function which either converts it into a host message and appends it to the host message buffer, or it causes an update to the NIU's state-of-the-world.

4.3 Data Flow. Data which the NIU operates on may be divided into two classes: Packets which are translated, and internal data which is part of the NIU overhead.

4.3.1 Packet Flow. Incoming Vehicle Appearance PDUS (VAPs) are placed into the NIU's State of the World (SOW) and will be processed during RVA. Other SIMNET PDU's are translated into a host-specific format and placed in the niu_to_host message buffer. After receiving the host_to_niu buffer the niu_to_host buffer is written to the host interface. This process is shown in Figure **TBD**.

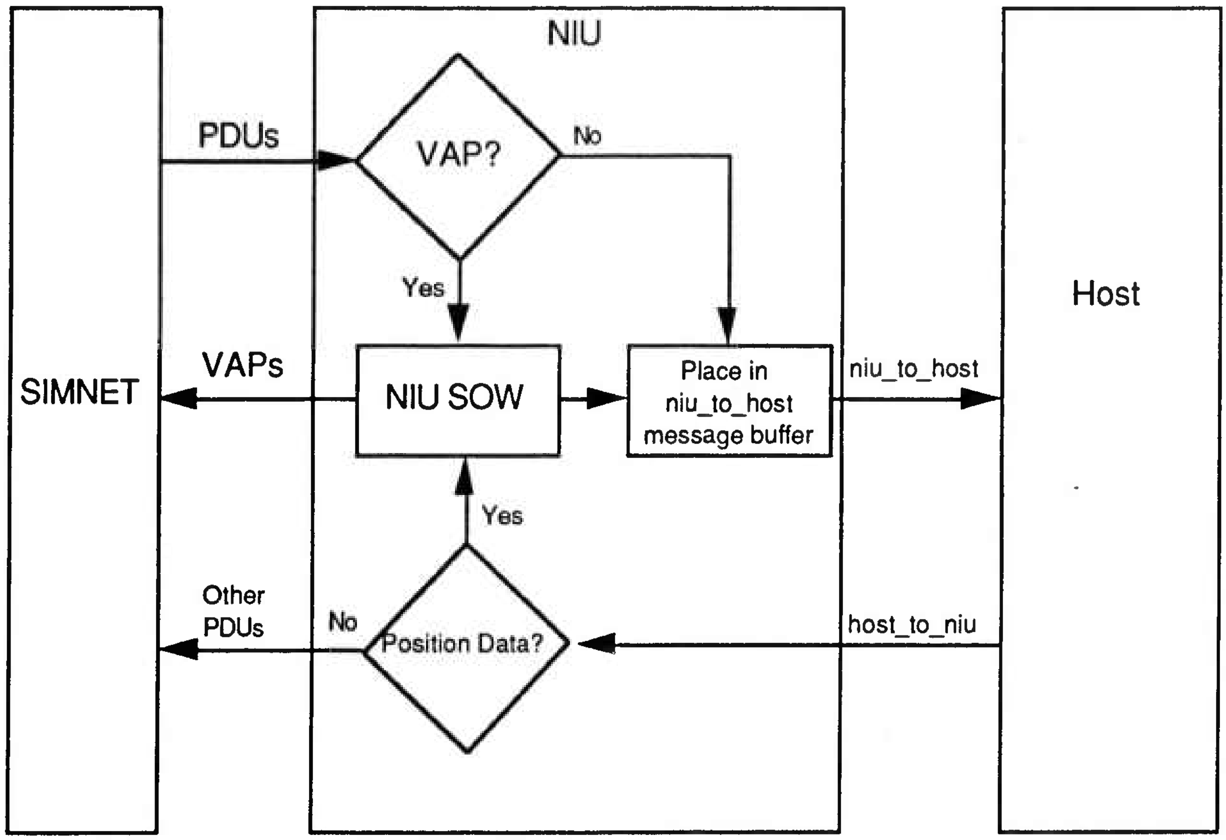


Figure TBD. Packet Flow through the NIU

4.3.2 Flow of Other Data.

4.4 Source Code Organization. Source code for the NIU is contained in libraries in two directory structures: NIU specific code is in the /simnet/niu/<host> structure, and the more general support code is contained in the /simnet/libsrc structure. Currently supported host interfaces are gci, cet, tg (ATES) and rrc (MDRC). NIU source is kept on seperate development machines (Sun Workstations). Figure **TBD** shows the control flow through the libraries. The individual libraries are discussed later on in this document.

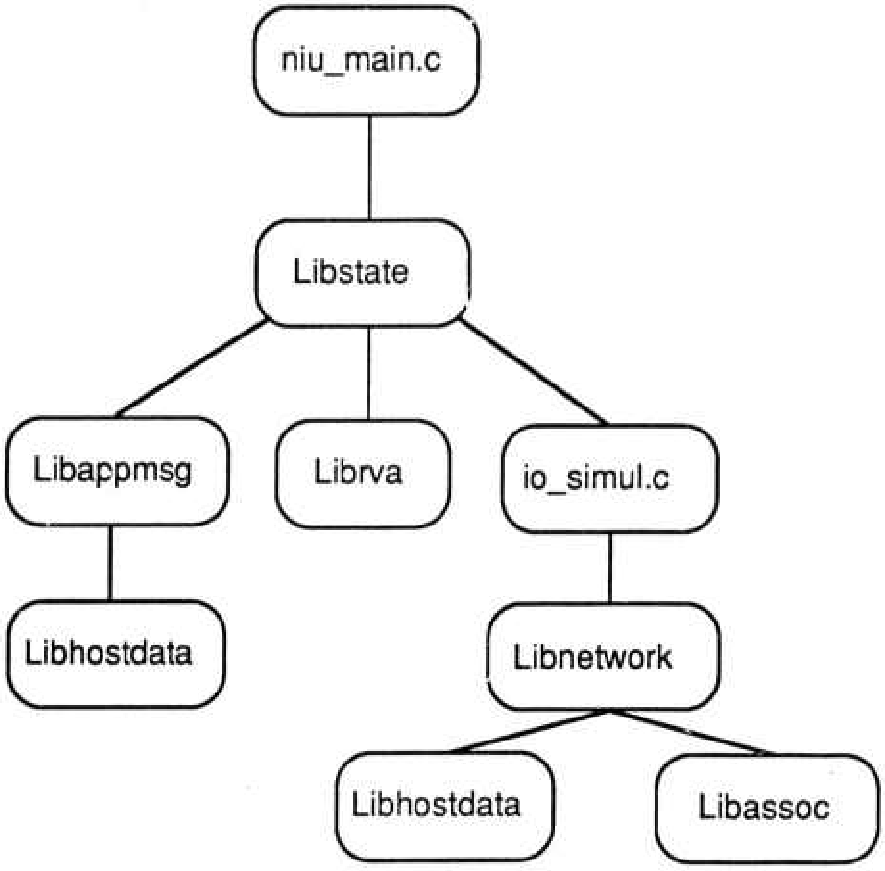


Figure TBD. Control Flow through Source Modules

4.4.1 NIU-Specific Libraries. The /simnet/niu/host tree contains source which is specific to the NIU. It is shown in Figure 7.

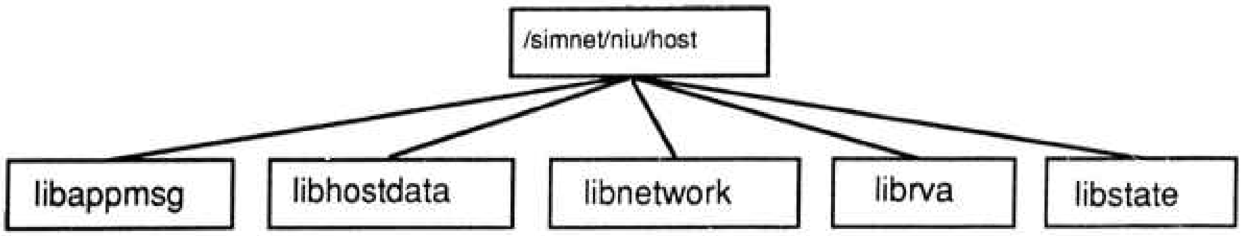


Figure 7. The NIU-Specific Source Tree

4.4.1.1 Libappmsg -- Host/NIU Interface. This contents of this library is dependant on the protocol being used to exchange data between the host and the NIU. Most protocols exchange data on vehicle position and attitude, weapon launches, and changes in the host's on-line state. The format of this data may differ from protocol to protocol.

In general, the NIU is presented with a buffer (host_to_niu) containing multiple messages. This buffer is processed by routines in libappmsg, event data will cause an immediate SIMNET PDU transmission, while appearance data is placed in the NIU's SOW. When the RVA models are updated the true position of the host vehicle is compared to the RVA model of the host vehicle. If the positions exceed the thresholds defined in nithresh.d a SIMNET VAP is transmitted. These PDUs are written immediately and are not buffered like messages to most hosts currently interfaced to the NIU.

4.4.1.2 Libhostdata -- The NIU's Host Database. Libhostdata is where information on the host is kept by the NIU. The exact contents of the database are host-dependant, and vary from the simple case of the host's frame rate and boiler-plates of all the supported SIMNET PDUs, to the more complicated case of data on multiple host vehicles.

In all cases, the data kept in the hostdata database is accessed through a set of access functions which read and write items in the database. No function outside of libhostdata should access the raw data-structures. This is not currently enforced, but may be in the future.

4.4.1.3 Libnetwork -- The NIU/SIMNET Interface. Libnetwork is a set of functions dedicated to supporting communications between the NIU and other SIMNET devices. When a packet is received each PDU is extracted and passed to a function which performs protocol specific actions based on the PDU's contents.

4.4.1.4 Librva -- Remote Vehicle Approximation. Librva contains functions which perform Remote Vehicle Approximation (RVA) on other simulator's vehicles. These vehicles are placed on priority lists which are defined in the file /simnet/niu/data/niuprlst.d on the NIU. These priority lists allow vehicles to be classified by range, type and team. Vehicles are then processed in decreasing order of priority. These lists were created mainly to support the BBN CIG, but are used to create the filter classes which control which PDUs are accepted by the NIU's ethernet adapter. A discussion of filtering can be found in section **4.4.2.4** of this document.

The RVA algorithm currently is a linear extrapolation of a vehicle's position and velocity vectors. The vehicle's position in 3-space is multiplied by the host's frame time and the vehicle's velocity along each axis. This gives the position which the vehicle would have moved to after one host frame has expired.

The bulk of librva is devoted to the maintenance of the priority lists. These lists also allow new lists to be created which contain vehicles new to the exercise since last frame, all vehicles which have been seen before, and all vehicles which have been destroyed this frame. These lists may be used in libappmsg to give special treatment to members of each list, dependant on the requirement of the host interface protocol. See section **4.4.1.1** for a description of libappmsg.

4.4.1.5 Libstate -- The Simulation State Machine. After niu_main.c has completed, control is transfered to the Simulation State Machine. This function as described in section **4.2** regulates the sequence in which the NIU performs various actions. Control never returns from the Simulation State Machine.

4.4.2 SIMNET Libraries. The /simnet/libsrc directory tree contains source which is used on multiple SIMNET devices and is not specific to any of them. Many of the libraries contain code which is not used in the NIU. The source tree is shown in Figure **TBD.**

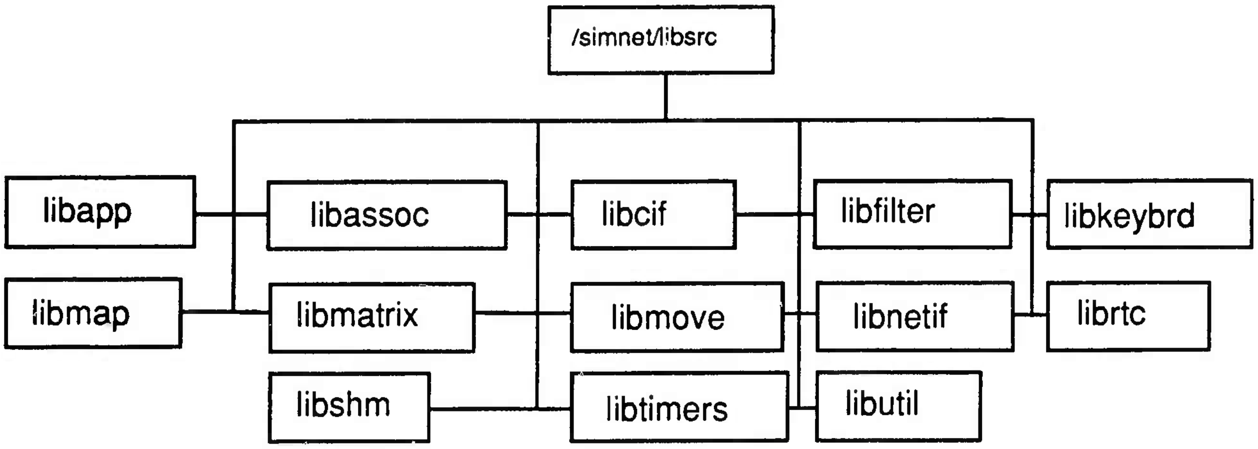


Figure TBD. SIMNET Source Tree

4.4.2.1 Libapp -- Threshold Logic. Libapp contains routines to read thresholds and test dead-reckoned positions against thresholds.

4.4.2.2 Libassoc -- The Association Layer. Libassoc contains routines to read and write packets to the network. It also contains channel maintenance, SIMNET transaction tracking, network initialization and multicast address support.

The Association layer is a transparent layer which exists between the application layer and the physical network. Each PDU has an association header prepended to it which is removed before the PDU is made available to the application layer. This is shown in Figure TBD.

Report 7102 describes the Association Layer as "a streamlined composite of the specific (OSI model) transport, session, and application layer services which are required by both the simulation and data collection protocols". Services provided by the presentation layer are not needed by SIMNET.

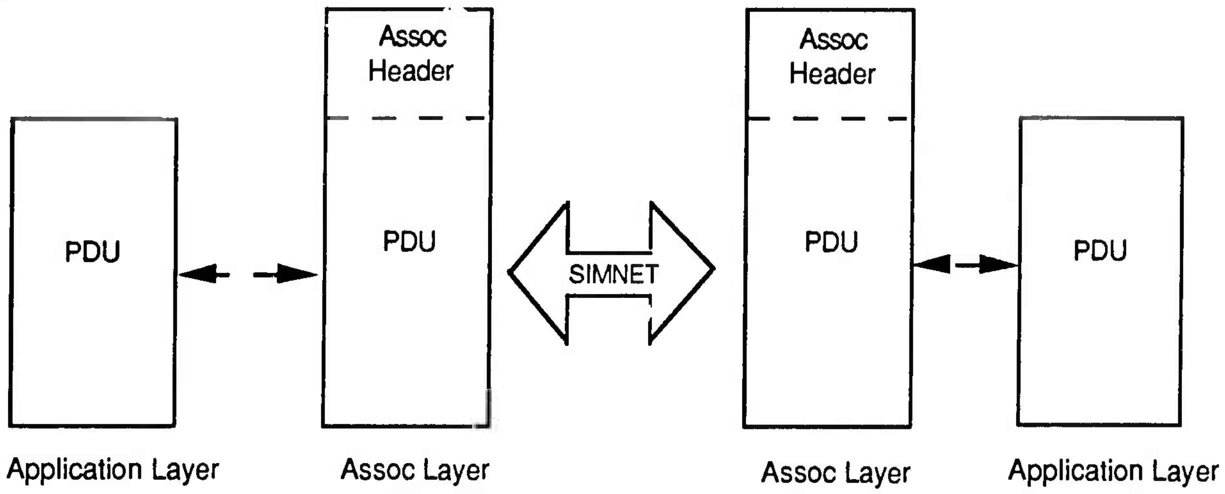


Figure TBD. Effect of the Association Layer

4.4.2.3 Libcif -- DR11W Driver. Libcif contains the driver for the DR11W communications interface. The DR11W is a high-speed parallel interface similar to an HSD interface.

4.4.2.4 Libfilter -- CMC Filter. Libfilter contains routines which interface to the downloaded portion of the CMC driver. This portion of the driver runs on the CMC card's local processor and talks to the 147-based portion of the driver through shared memory. Libfilter is used to initialize the downloaded portion by setting up filtering conditions which the downloaded driver uses to determine if a PDU should be passed up to the 147's CMC driver, this allows the CMC card to offload the filtering so that the 147 is free to perform other tasks. The relationshp between both halves of the CMC driver is shown in Figure **TBD.**

The filter conditions are taken from the priority lists which are in /simnet/niu/data/niuprlst.d on the NIU's disk and are written to the CMC card in rva_pr_init() during NIU initialization.

This library assumes that SIMNET is being serviced by a CMC card, and uses shared memory to communicate with the downloaded portion of the CMC driver which is running on the CMC card.

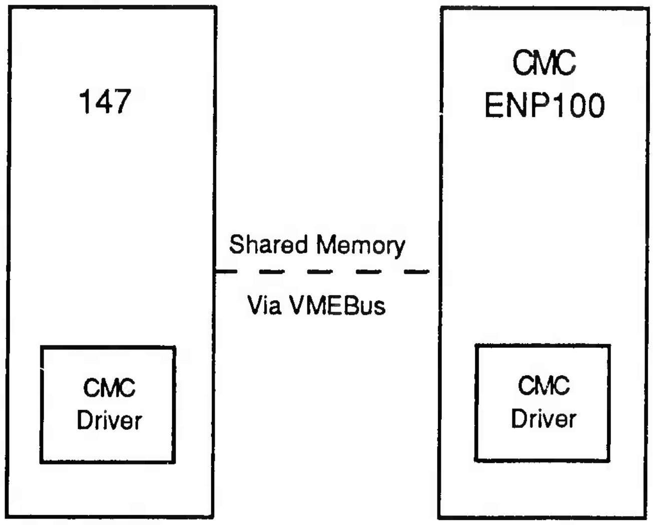


Figure TBD. Division of CMC Driver Halves

<u>4.4.2.5 Libkeybrd -- Keyboard Device Interface</u>. Libkeybrd contains open, close, read, write, and reset routines for the keyboard device. The keyboard may be used during NIU operation to display statistics on NIU operation, as well as starting and stopping the NIU. See Appendix **TBD** for a complete listing of keyboard commands supported.

<u>4.4.2.6 Libmap -- Read a Culture Map</u>. Libmap is used to support the BBN CIG. It reads in a culture map containing vehicles, buildings, lifeforms, etc. Libmap is not currently used in the NIU.

4.4.2.7 Libmatrix -- Matrix Math Library. Libmatrix contains routines which perform math operations on quaternions, direction cosine matrices, vectors and other matrix types. Most of these routines are not used in the NIU.

4.4.2.8 Libmove -- Data Movement Library. Libmove contains routines which move blocks of data in 8, 16, and 32 bit increments. These are useful to obtain greater speed during block copies, and since some cards require accesses to be made on a specified alignment boundary.

4.4.2.9 Libnetif -- Low-Level Network Interface. Libnetif contains a generic ethernet driver which runs on top of the drivers supplied with VRTX. This driver translates I/O requests into requests directed at the actual ethernet driver. For example: a read request is turned into either a CMC read request or a 147 read request depending on which card was specified. This allows application code to be written independant of the network interface which is actually being used. Figure **TBD** shows the relationship between Libnetif, a CMC driver, and a 147 ethernet driver.

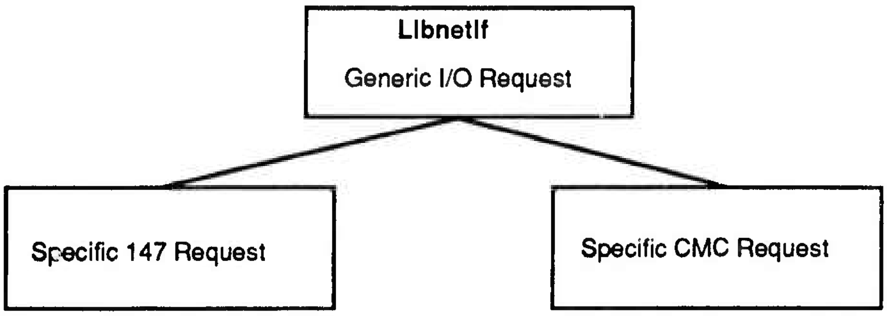


Figure TBD. Libnetif Operation

4.4.2.10 Librtc -- Real Time Clock Library. Librtc contains routines which start, stop, read and clear the real time clock. It also maintains separate timers which can be started and stopped independantly of each other. These are used for determining how much time is spent in a given section of code.

4.4.2.11 Libshm -- Shared Memory Library. Libshm contains routines which manage shared memory segments. These are not used on the NIU.

<u>4.4.2.12 Libtimers -- Virtual Timers Library</u>. Libtimers contains routines which manage virtual timers. These timers allow a function to be called after a specified amount of time has expired. The alarm may then be rescheduled, or can be cleared. These are not used.

<u>4.4.2.13 Libutil -- Misc Utility Functions</u>. This library contains routines which do not belong in other libraries, such as clearing the console's screen, creating a core image on disk, some new matrix routines, and a number of routines to display data-structures.

SECTION 5. MECHANICAL/ELECTRICAL IMPLEMENTATION

5.1 The NIU is contained in an aluminum chassis designed for installation into a 19" rack. The backplane used for board interconnection conforms to the VMEbus Specification Rev C.1. The chassis is referred to as a Double Height VMEbus sub-rack in the VMEbus specification. See Figure 5.1 for specification of individual board locations.

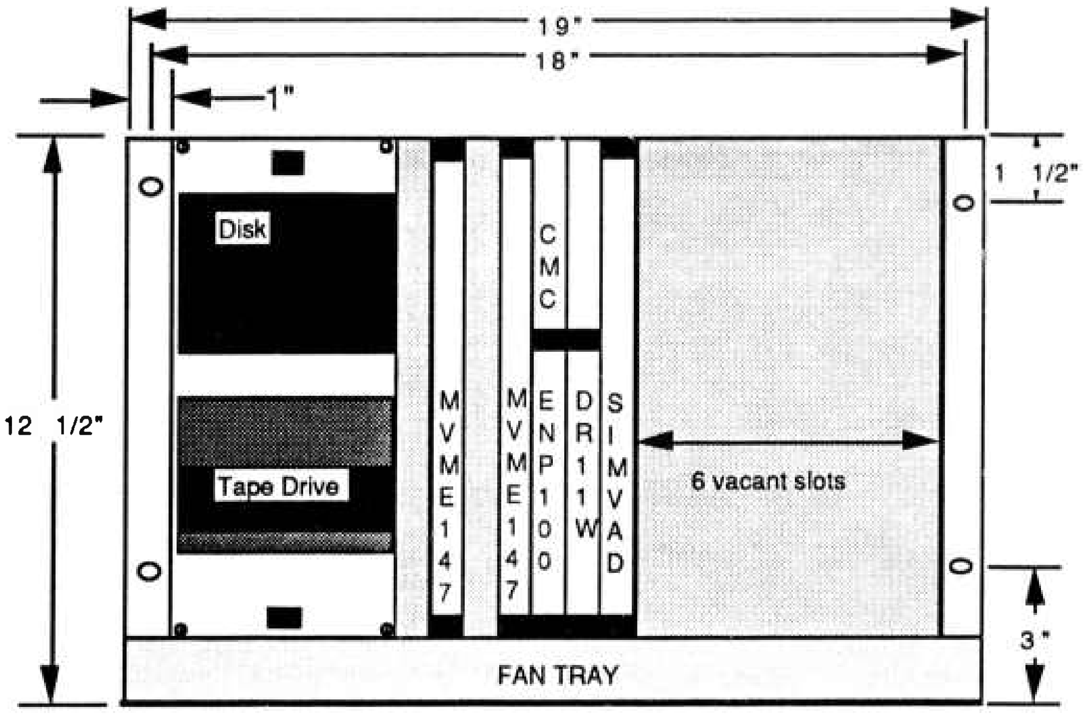


Figure 5.1 Front View - Network Interface Unit

5.1.1 Electrical Specifications - The NIU has only one power supply. This power supply provides energy to all boards within the chassis and to the Audio Interface Adapter if attached. The electrical specifications for the power supply are :

Power Supply:	TODD Products Corporation Model MAX-754-1205P
Input:	90-132 Vac, 47-63Hz. Internally fused for 15A.
Outputs:	+5V dc @120A
	+12V dc @12A/20A peak
	-12V dc @ 10.0A
	+5.2V dc @ 2.0A
Adjustability:	All outputs adjustable +/-5% minimum.

Temperature Range: 0-50 degrees C full ratings.
Cooling: Built-in ball bearing fan.
See Appendix **TBD** for complete specifications.

5.1.2 Mechanical Specifications - The chassi is received already assembeld, the only remaining assembly required is for board and mass storage assembly installation.

5.1.2.1 MVME147-1 Installation and Configuration - The MVME147 should have jumpers placed on the following:
- **J3** pins 2 &4, 3 &5, 6 &8, 13 &15, 14 &16
- **J4** pins 2 &4, 3 &5, 6 &8, 13 &15, 14 &16
- **J5** pins 1 &2
- **J6** pins 1 &2
- **J7** pins 2 &4
- **J8** pins 5 &6
- **J9** pins 2 &3
- **J10** pins 1 &2

The MVME712M board used for support should have jumpers across the pins as follows:
- **J1** pins 1&2, 3&4, 5&6, 7&8, 9&10, 11&12, 13&14
- **J14** pins 1&2, 3&4, 5&6, 7&8, 9&10, 11&12, 13&14
- **J17** pins 1&2, 3&4, 5&6, 7&8, 9&10, 11&12, 13&14
- **J19** pins 1&2, 3&4, 5&6, 7&8, 9&10, 11&12, 13&14

All other pins should remain unjumpered.

There are no jumpers for consideration on the daughter board of the MVME147.